UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Corso di
FONDAMENTI DI PROGRAMMAZIONE

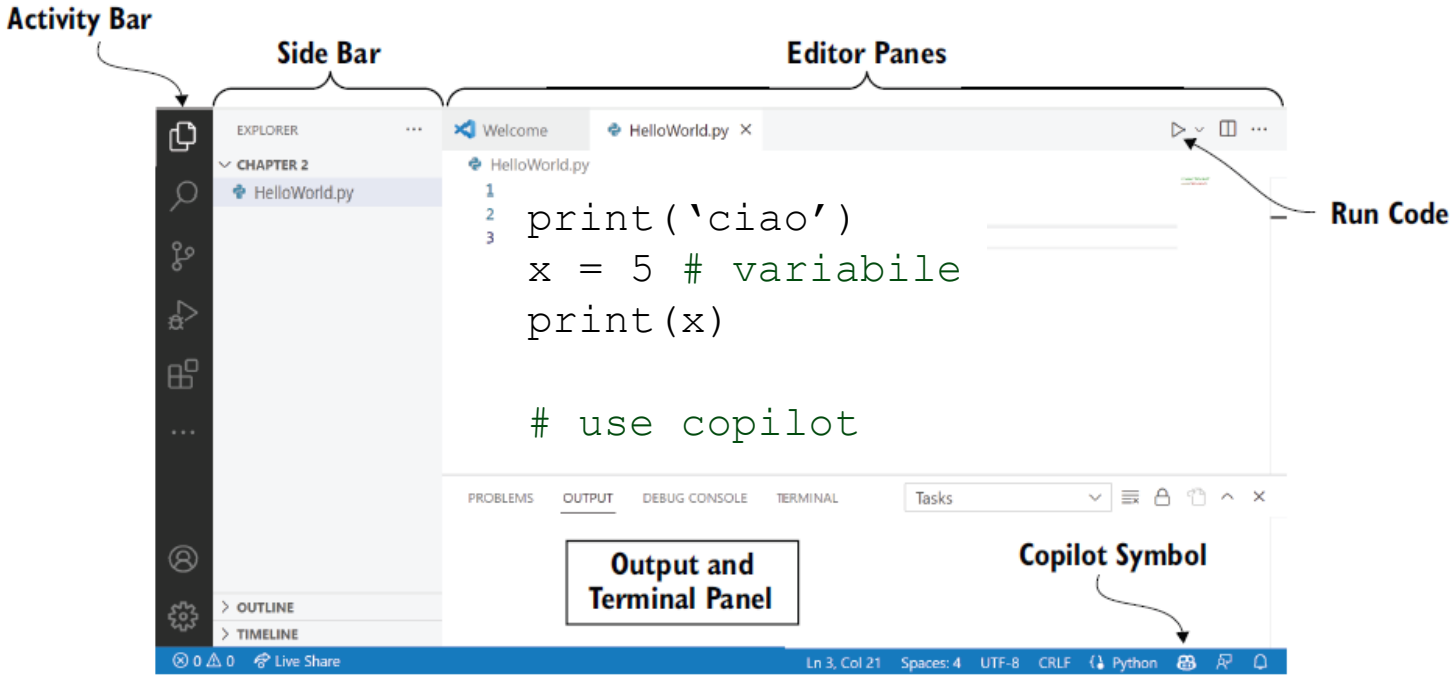
Introduzione a Python

Prof. Marco Mamei

Python

Python3
IDE (VSCode)
Copilot

Script (.py)
Documenti interattivi (.ipynb)



Esempi di analisi di dati

Quarterbacks are responsible for throwing the football. **We want to print how many yards each quarterback passed for in that time period**. To include only the quarterbacks, we'll need to tell Copilot to limit our results to just players whose *Position* (the third column) is *QB* (stands for Quarterback). The number of passed yards is in the 8th column (pass_yds).

Figure 2.5 The first few columns and rows of the nfl_offensive_stats.csv dataset

https://mng.bz/86pw

Considerazioni

- Copilot (e gli LLM in generale) sono molto potenti… non abbiamo scritto codice

- **E’ importante riuscire a dividere il programma in sotto task**
  - E’ più facile che copilot sbagli un programma complesso che tanti sotto-task semplici

- Abbiamo ancora bisogno di capire il codice a un certo livello
  - Utile sia per indirizzare meglio copilot tramite i prompt
  - Sia per individuare errori

- E’ fondamentale testare il programma generato

Funzioni

- Qual è un (sotto)task che posso richiedere a copilot? Se il task è troppo grande e articolato è probabile che copilot non riesca a generare il codice e che cmq questo sia incomprensibile

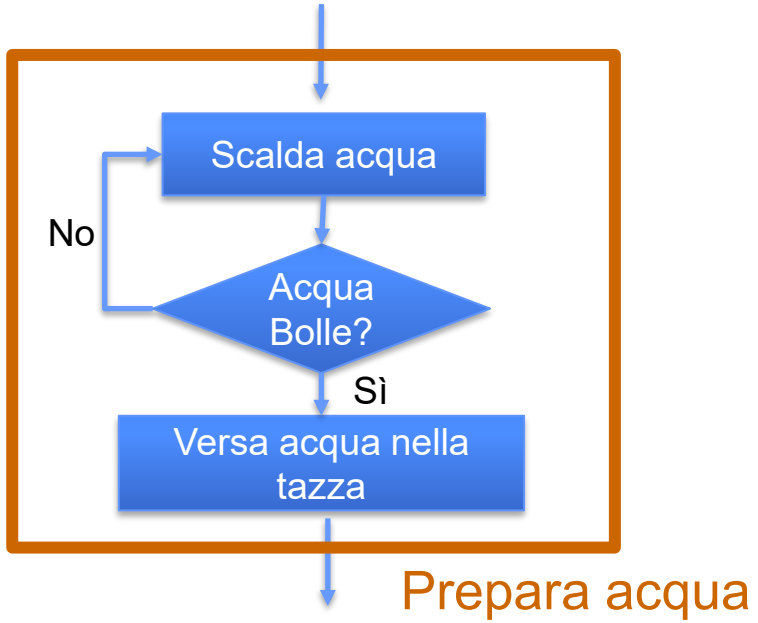
- E' una domanda importante per copilot, ma più in generale dividere un compito in sotto task più semplici è alla base dell'ingegneria

- **In python le funzioni permettono di definire un task ben preciso che può essere eseguito una o più volte.**

- Una funzione svolge un compito ben preciso ed è abbastanza semplice da comprendere

Esempio


Inizio
Prepara acqua
Aggiungi bustina di tè
Attendi alcuni minuti
Zucchero ?
Sì
Aggiungi zucchero
No
Fine


Scalda acqua
No
Acqua Bolle?
Sì
Versa acqua nella tazza
Prepara acqua

Esempio

R	M	E	L	L	L	D	I	L	A	Z	K
B	F	W	H	F	M	O	Z	G	L	Z	C
B	D	T	U	C	N	G	S	L	S	H	A
Y	Y	O	F	U	N	C	T	I	O	N	T
F	A	H	S	I	L	T	A	S	K	O	C
H	N	H	J	O	H	E	L	L	O	C	A
Y	F	M	P	I	P	W	L	B	T	R	J
L	N	S	J	N	E	Z	Y	Z	Z	I	T

Find the following hidden words in the puzzle:

CAT	FUNCTION	TASK
DOG	HELLO	

- At a high level, your task is “find all the words in the word search.”

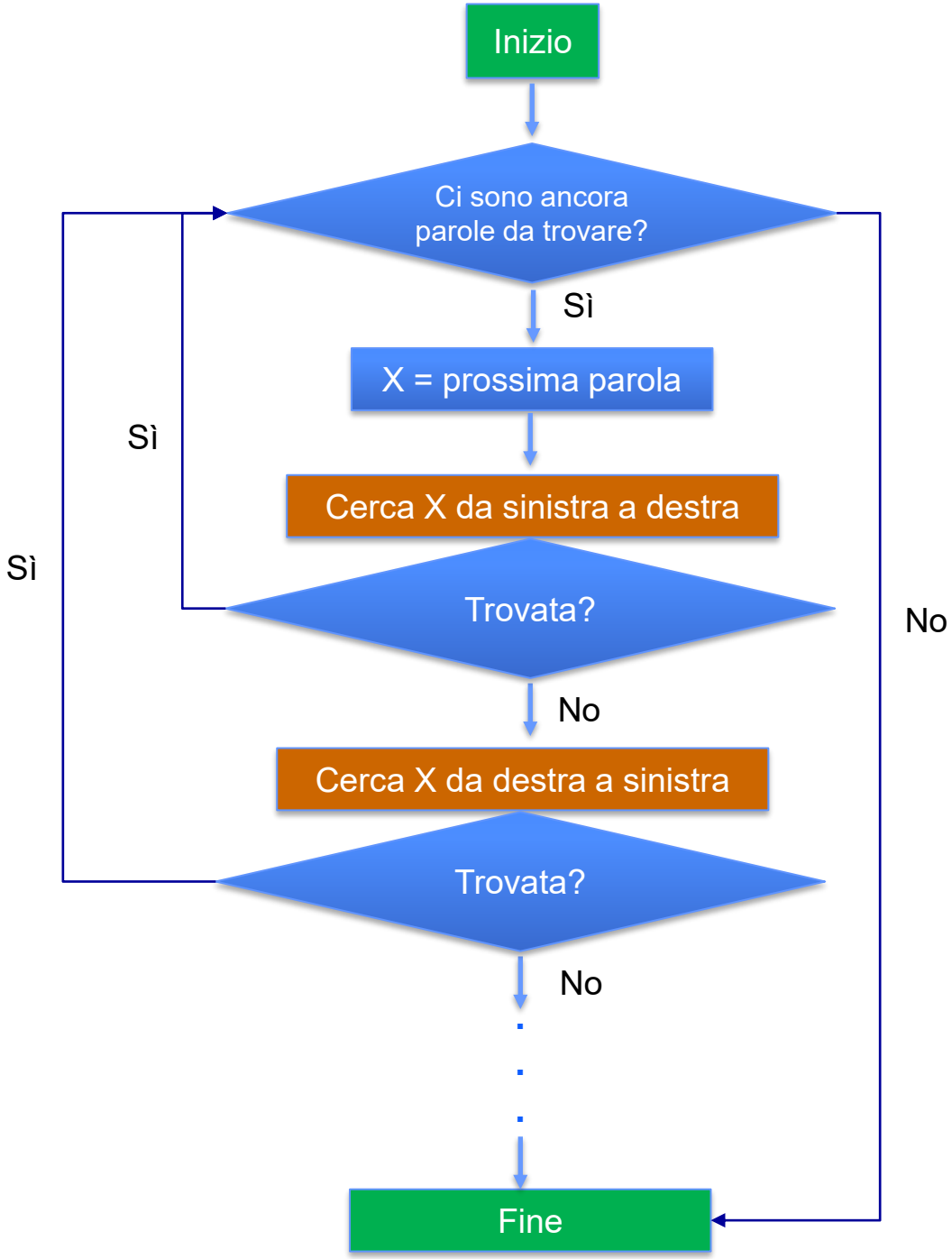
Example

- Try working on the problem right now for a couple minutes. ***How did you start? How did you break down the overall task to make it more achievable?***

- One thing you might do is say, “OK, finding every word is a big task, but a smaller task is **just finding the first word** (CAT). Let me work on that first!” **This is an example of taking a large task and breaking it into smaller tasks**. To solve the entire puzzle, then, you could repeat that smaller task for each word that you need to find.

- Now, how would we find an individual word, such as CAT? Even this task can be broken down further to make it easier to accomplish. For example, we could break it into four tasks: search for CAT from left to right, search for CAT from right to left, search for CAT from top to bottom, and search for CAT from bottom to top.

- Taking a large problem and dividing it into smaller tasks is called ***problem decomposition***


Inizio
Ci sono ancora parole da trovare?
Sì
X = prossima parola
Cerca X da sinistra a destra
Trovata?
Sì
No
Cerca X da destra a sinistra
Trovata?
Sì
No
No
Fine

Funzioni

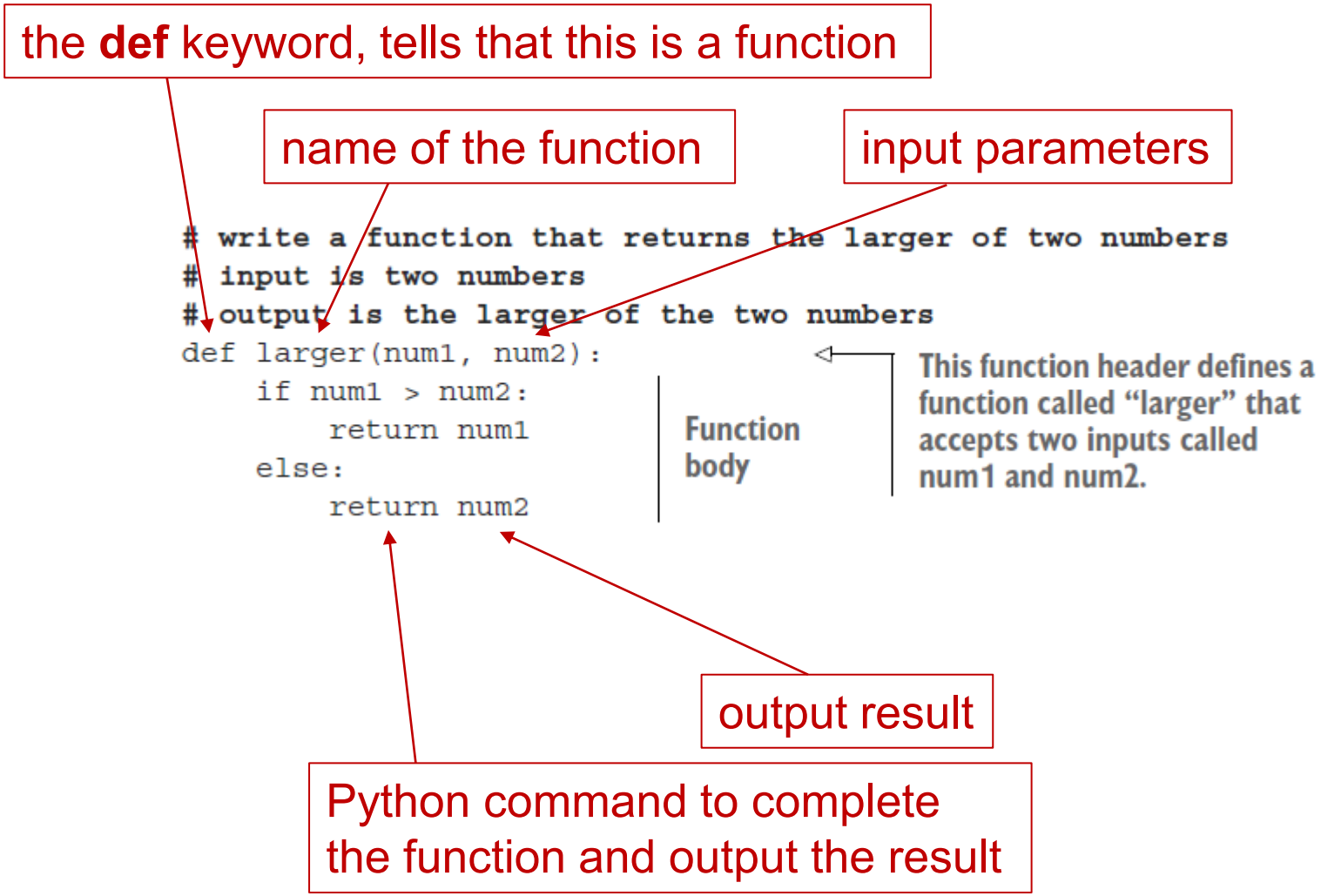
- The origin of the name *function* goes back to math where functions define the output of something based on an input: $f(x) = x^2$
- As programming functions also have expected output for a particular input.
- As programmers, we also like to think of functions as promises or contracts. If there is a function called larger, and we're told that it takes two numbers and gives us the larger of the two, we have faith that when we give the function the numbers 2 and 5, it will return the answer of 5. We don't need to see how that function works to use it, any more than we need to know how the mechanics of a car works to use the brake pedal.
- Every function in Python has a *function header* (also called a *function signature*), which is the first line of code of the function. Given their ubiquitous nature, we'll want to read and write function headers

```
# write a function that returns the larger of two numbers
# input is two numbers
# output is the larger of the two numbers
def larger(num1, num2):
    if num1 > num2:
        return num1
    else:
        return num2
```

Function body

This function header defines a function called "larger" that accepts two inputs called num1 and num2.

Funzioni


the **def** keyword, tells that this is a function
name of the function
input parameters
# write a function that returns the larger of two numbers
# input is two numbers
# output is the larger of the two numbers
def larger(num1, num2):
if num1 > num2:
return num1
else:
return num2
Function body
This function header defines a function called "larger" that accepts two inputs called num1 and num2.
output result
Python command to complete the function and output the result

Docstrings

- **Docstrings explain function behavior**, *Docstrings* are how Python functions are described by programmers. They follow the function header and begin and end with three double quotation marks.

- By writing the header and docstring, you'll make it easier for Copilot to generate the right code. **In the header, you'll be the one deciding on the name of the function and providing the names of each parameter that you want the function to use**. After the function header, you'll provide a docstring that tells Copilot what the function does.

```
def larger(num1, num2):
    """
    num1 and num2 are two numbers.

    Return the larger of the two numbers.
    """
    if num1 > num2:
        return num1
    else:
        return num2
```

Docstring description of the function

Using a function

- The way to use a function is to *call* it. Calling a function means to invoke the function on specific values of parameters. These parameter values are called arguments

- Each value in Python has a ***type***, and we need to take care to give values of the proper type. For example, that larger function is expecting two numbers; it might not work as expected if we supply inputs that aren't numbers.
- When we call a function, it runs its code and returns its result. We need to capture that result so that we can use it later; otherwise, it will be lost. To capture a result, we use a ***variable***, which is just a name that refers to a value.

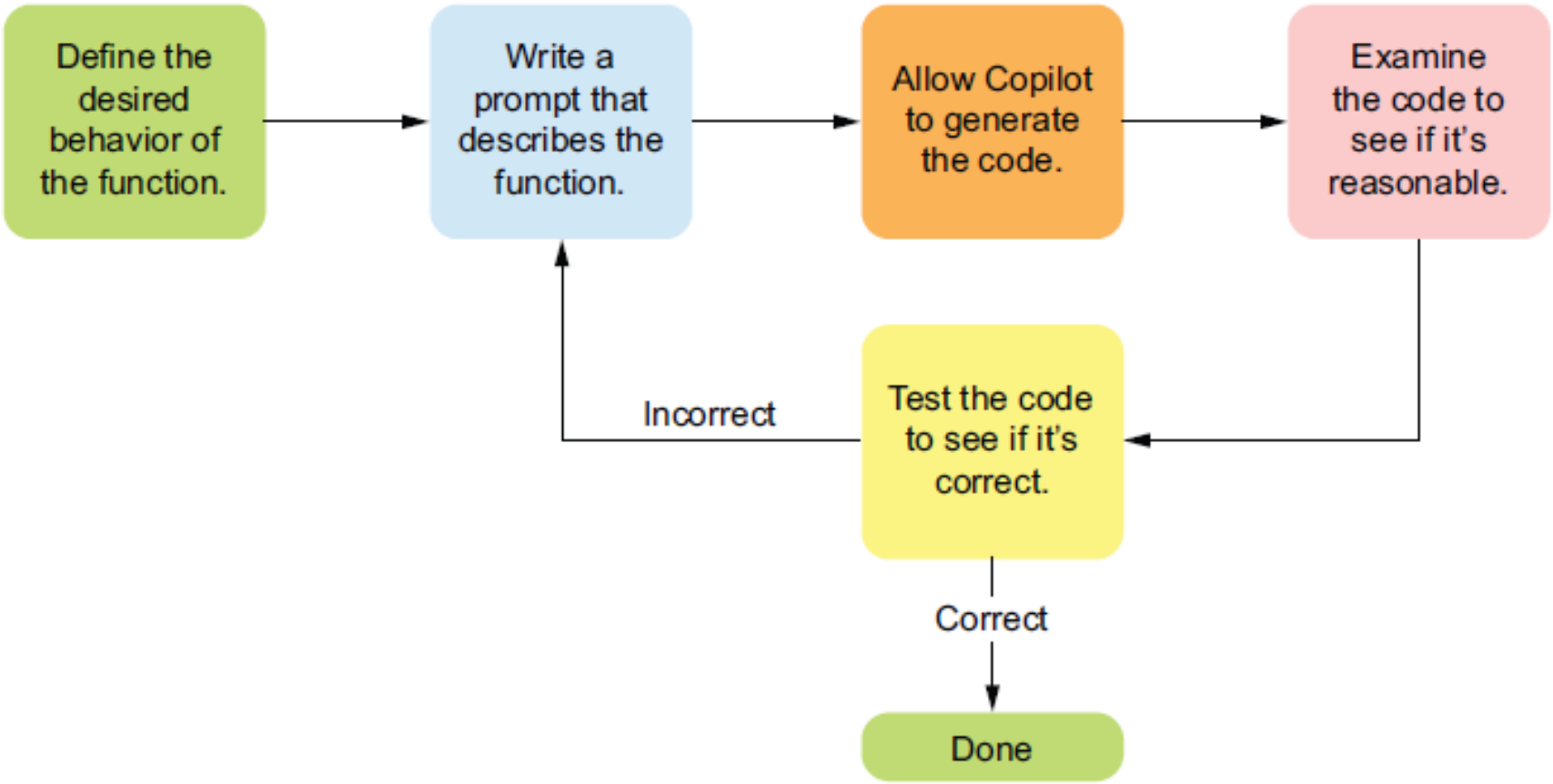
```
# call the larger function with the values 3 and 5
# store the result in a variable called result
# then print result
result = larger(3, 5)
print(result)
```

Calls the larger function with the values 3 and 5 as inputs and stores the result

- The general format for a function call is

```
function_name(argument1, argument2, argument3,... )
```

Come creare funzioni con copilot



Esempio

Dan's stock pick. Dan is an investor in a stock called AAAPL. He purchased 10 shares for $15 each. Now, each of those shares is worth $17. Dan would like to know how much money he has made on the stock.

Remember that we want to make our **function as general as possible**. A useful general function here would take three parameters, all of which are numbers. The first parameter is the number of shares purchased, the second is the share price when the shares were purchased, and the third is the current share price.

Let's call this function **money_made**, because it's going to determine how much money we've made or lost on the stock. In general, you want to name your function as an action word or words that describe what your function is doing. With that, we have enough to write the function header:

```
def money_made(num_shares, purchase_share_price, current_share_price):
```

Now, we need a docstring. In the docstring, we need to explain what each parameter is for by using its name in a sentence.

```
def money_made(num_shares, purchase_share_price, current_share_price):
    """
    num_shares is the number of shares of a stock that we purchased.
    purchase_share_price is the price of each of those shares.
    current_share_price is the current share price.

    Return the amount of money we have earned on the stock.
    """
```

Esempio

Leo is signing up for a new social network website. He wants to make sure that his password is strong.

Unlike our previous functions, we're not dealing with numbers here. The parameter, the password to check, is text. And the return value is supposed to indicate some yes/no result. We need new types!

- The Python type for text is called a *string*.
- The Python type for a yes/no result is called a *Boolean* or *bool*. A bool has only two values: True or False.

```
def is_strong_password(password):
    """
    A strong password has at least one uppercase character,
    at least one number, and at least one special symbol.

    Return True if the password is a strong password, False if not.
    """
```

```
def get_strong_password():
    """
    Keep asking the user for a password until it's a strong
    password, and return that strong password.
    """
```

Esempio

- ***Scrabble scoring.*** One of Dan's favorite board games is Scrabble.

```
def num_points(word):
    """
    Each letter is worth the following points:
    a, e, i, o, u, l, n, s, t, r: 1 point
    d, g: 2 points
    b, c, m, p: 3 points
    f, h, v, w, y: 4 points
    k: 5 points
    j, x: 8 points
    q, z: 10 points

    word is a word consisting of lowercase characters.
    Return the sum of points for each letter in word.
    """
```

- Let's continue with the Scrabble theme. Suppose that Dan has a bunch of words that he can make right now, but he doesn't know which one will give him the most points.

```
def best_word(word_list):
    """
    word_list is a list of words.

    Return the word worth the most points.
    """
```

Vantaggi dell’uso delle funzioni

- Ridurre il carico cognitivo
- Evitare ripetizioni
- Semplificare il testing e individuare errori più facilmente
- Migliorare la leggibilità del codice

Chiamate di funzioni

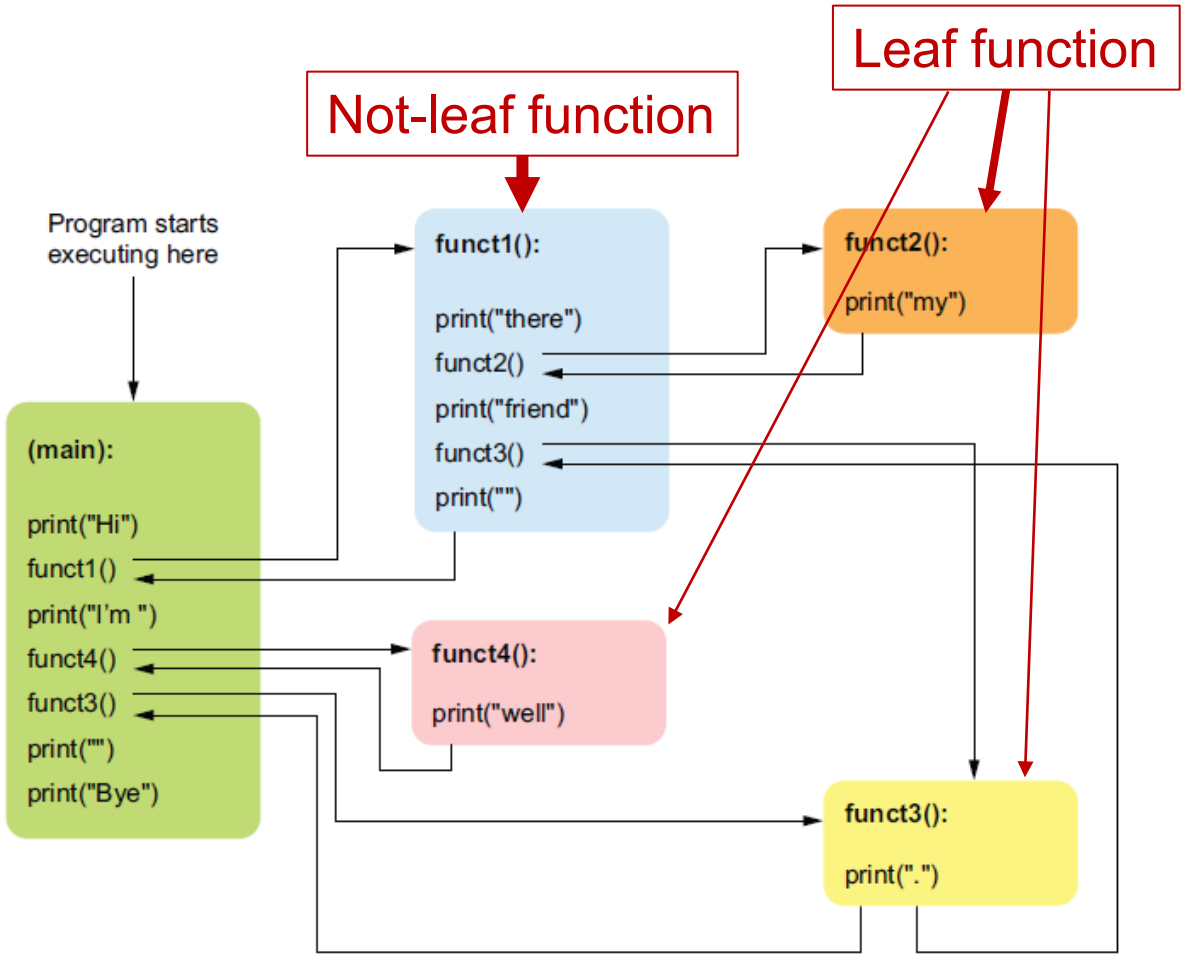
```
def funct1():
    print("there")
    funct2()
    print("friend")
    funct3()
    print("")

def funct2():
    print("my")

def funct3():
    print(".")

def funct4():
    print("well")


print("Hi")
funct1()
print("I'm")
funct4()
funct3()
print("")
print("Bye.")
```



Come decidere quali funzioni creare?

- ***One clear task to perform***—A leaf function might be something like “compute the volume of a sphere,” “find the largest number in a list,” or “check to see if a list contains a specific value.” Nonleaf functions can achieve broader goals, like “update the game graphics” or “collect and sanitize input from the user.” Nonleaf functions should still have a particular goal in mind, but they are designed knowing that they will likely call other functions to achieve their goal.

- ***Clearly defined behavior***—The task “find the largest number in a list” is clearly defined. In contrast, the task “find the best word in the list” is poorly defined as stated: What is the “best” word?

- ***Short in number of lines of code***—We’ve heard different rules over the years for the length of functions, informed by different company style guidelines. The lengths we’ve heard vary from 12 to 20 lines of Python code as the maximum number of lines

- ***General value over specific use***—A function that returns the number of values in a list that are greater than 1 might be a specific need for a part of your program, but there’s a way to make this better. The function should be rewritten to return the number of values in the list that are greater than another parameter.

- ***Clear input and output***—You generally don’t want a lot of parameters. That doesn’t mean you can’t have a lot of input, though. A single parameter could be a list of items, as in our best_word function in section. You can only return one thing, but again, you can return a list so you aren’t as limited as it may appear. But if you find yourself writing a function that sometimes returns a list, sometimes returns a single value, and sometimes returns nothing, that’s probably not a good function.

Examples

- *Good*
  - *Compute the volume of a sphere*
  - *Find the largest number in a list*
  - *Check whether a list contains a specific value*
  - *Print the state of the checkers game*

- *Bad*
  - *Request a user's tax information and return the amount they owe this year*
  - *Identify the largest value in the list and remove that value from the list*
  - *Return the names of the quarterbacks with more than 4,000 yards of passing in the dataset*
  - *Determine the best movie of all time*
  - *Play Call of Duty*